Deep Generative Models: Beyond GANs

Raunak Kumar

University of British Columbia

MLRG, 2017 Winter Term 1

November 28, 2017

Overview

1. Problem Setup
2. Density Estimation using Real NVP
   - Outline
   - Change of Variable
   - Coupling Layers
   - Experiments
   - Summary

Problem Setup

Density Estimation

- Goal: Construct a generative probabilistic model.

Density Estimation

- Goal: Construct a generative probabilistic model.
- Data is high dimensional and highly structured.
- Challenge: Build complex yet trainable models.

Density Estimation

- Goal: Construct a generative probabilistic model.
- Data is high dimensional and highly structured.
- Challenge: Build complex yet trainable models.
- We've seen VAEs and GANs over the past month.
- Today we will talk about *Density Estimation using Real NVP* [1].
- If we have time, I will briefly go over *Pixel Recurrent Neural Networks* [2].

Density Estimation using Real NVP

Gameplan

1. Want to use maximum likelihood estimation.
2. Write the likelihood of data in a form that is easy to compute.
3. Compute using deep neural networks.

Likelihood

- Goal: Learn $P_X(x)$ using maximum likelihood estimation.

Likelihood

- Goal: Learn $P_X(x)$ using maximum likelihood estimation.
- But how do we compute likelihood of the data?
- Can we write $P_X(x)$ in a way that is easy to compute?

Likelihood

- Goal: Learn $P_X(x)$ using maximum likelihood estimation.
- But how do we compute likelihood of the data?
- Can we write $P_X(x)$ in a way that is easy to compute?

  $\rightarrow$ Use *change of variable* formula!

Change of Variable

Given:

- observed data variable $x \in X$,
- latent variable $z \in Z$ with prior probability distribution P_Z,
- a *bijection* $f : X \to Z$ with its inverse denoted as g,

Change of Variable

Given:

- observed data variable $x \in X$,
- latent variable $z \in Z$ with prior probability distribution P_Z,
- a *bijection* $f : X \to Z$ with its inverse denoted as g,

the change of variable formula defines a distribution on x as

$$P_X(x) = P_Z(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| . \tag{1}$$

Change of Variable

Given:

- observed data variable $x \in X$,
- latent variable $z \in Z$ with prior probability distribution P_Z,
- a *bijection* $f : X \to Z$ with its inverse denoted as g,

the change of variable formula defines a distribution on x as

$$P_X(x) = P_Z(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| . \quad (1)$$

Proof sketch for 1D case:

1. Express the CDF of X in terms of an integral over Z using the bijection f.
2. Compute the density of X using the Fundamental Theorem of Calculus and Chain Rule.

Good News

Sampling

- Draw $z \sim P_Z$, and generate a sample $x = f^{-1}(z) = g(z)$.

Inference

- $P_X(x) =$ product of $P_Z(f(x))$ and its Jacobian determinant.

Bad News

Data and latent spaces are both very high-dimensional. So:

- Jacobian matrix is huge!
- Computing the Jacobian is expensive.
- Computing its determinant is expensive.

But . . .

Observe:

- Only need the determinant of the Jacobian, not the Jacobian itself.
- Determinant of a triangular matrix is the product of its diagonal terms.

But . . .

Observe:

- Only need the determinant of the Jacobian, not the Jacobian itself.
- Determinant of a triangular matrix is the product of its diagonal terms.

  $\rightarrow$ Design bijective f such that its Jacobian is triangular.
  $\rightarrow$ Use change of variable formula.
  $\rightarrow$ Train using maximum likelihood estimation.
  $\rightarrow$ Once we learn f, we can do sampling and inference.

But . . .

Observe:

- Only need the determinant of the Jacobian, not the Jacobian itself.
- Determinant of a triangular matrix is the product of its diagonal terms.
  - $\rightarrow$ Design bijective f such that its Jacobian is triangular.
  - $\rightarrow$ Use change of variable formula.
  - $\rightarrow$ Train using maximum likelihood estimation.
  - $\rightarrow$ Once we learn f, we can do sampling and inference.

How do we design such an f???

But . . .

Observe:

- Only need the determinant of the Jacobian, not the Jacobian itself.
- Determinant of a triangular matrix is the product of its diagonal terms.
  - → Design bijective f such that its Jacobian is triangular.
  - → Use change of variable formula.
  - → Train using maximum likelihood estimation.
  - → Once we learn f, we can do sampling and inference.

How do we design such an f???
Why no deep neural networks yet???

Designing f

- Composition of bijections is a bijection.

Designing f

- Composition of bijections is a bijection.
  $\rightarrow$ Build f by composing a sequence of simple bijections.
- Each such simple bijection is called an *affine coupling layer*.

Coupling Layer

Given $x \in \mathbb{R}^D$ and $d < D$, the output of an affine coupling layer, y, is:

$$y_{1:d} = x_{1:d}, \tag{2}$$

$$y_{d+1:D} = x_{d+1:D} \circ \exp\left(s(x_{1:d})\right) + t(x_{1:d}), \tag{3}$$

where $s, t : \mathbb{R}^d \to \mathbb{R}^{D-d}$ are *arbitrary* functions.
(arbitrary means deep convolutional neural networks . . .)

Coupling Layer

Given $x \in \mathbb{R}^D$ and $d < D$, the output of an affine coupling layer, y, is:

$$y_{1:d} = x_{1:d}, \qquad (2)$$

$$y_{d+1:D} = x_{d+1:D} \circ \exp\left(s(x_{1:d})\right) + t(x_{1:d}), \qquad (3)$$

where $s, t : \mathbb{R}^d \to \mathbb{R}^{D-d}$ are *arbitrary* functions.
(arbitrary means deep convolutional neural networks . . .)

Since we are interested in sampling and inference, what about the inverse and the Jacobian determinant?

Inverse

- f:

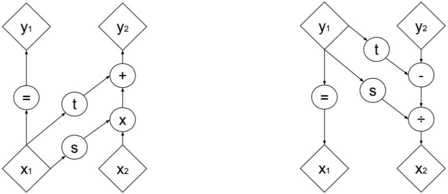
$$
\begin{aligned}
y_{1:d} &= x_{1:d}, \\
y_{d+1:D} &= x_{d+1:D} \circ \exp\left(s(x_{1:d})\right) + t(x_{1:d}),
\end{aligned}
$$

- g:

$$
\begin{aligned}
x_{1:d} &= y_{1:d}, && (4) \\
x_{d+1:D} &= \left(y_{d+1:D} - t(y_{1:d})\right) \circ \exp\left(-s(y_{1:d})\right) && (5)
\end{aligned}
$$

- Sampling is as efficient as inference.
- No need to invert s or t.

Inverse



(a) Forward propagation (b) Inverse propagation

Figure 2: Computational graphs for forward and inverse propagation. A coupling layer applies a simple invertible transformation consisting of scaling followed by addition of a constant offset to one part $\mathbf{x}_2$ of the input vector conditioned on the remaining part of the input vector $\mathbf{x}_1$. Because of its simple nature, this transformation is both easily invertible and possesses a tractable determinant. However, the conditional nature of this transformation, captured by the functions s and t, significantly increase the flexibility of this otherwise weak function. The forward and inverse propagation operations have identical computational cost.

Jacobian

- The Jacobian has the following form:

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}(\exp s(x_{1:d})) \end{bmatrix}. \tag{6}$$

- Determinant: $exp\left(\sum_j s(x_{1:d})_j\right)$. *Easy* to compute.
- No need to compute Jacobian of s or t.

Masked Convolution

Using a binary mask b, the output of the coupling layer can be written as

$$y = b \circ x + (1 - b) \circ (x \circ \exp(s(b \circ x)) + t(b \circ x)). \quad (7)$$

We will see examples of masks shortly.

Combining Coupling Layers

- A coupling layer leaves some components of the input unchanged.

Combining Coupling Layers

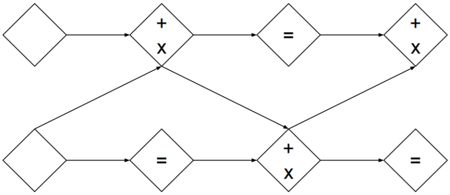
- A coupling layer leaves some components of the input unchanged.

  $\rightarrow$ Compose such layers in an alternating pattern.

Combining Coupling Layers

- A coupling layer leaves some components of the input unchanged.

  $\rightarrow$ Compose such layers in an alternating pattern.



(a) In this alternating pattern, units which remain identical in one transformation are modified in the next.

Combining Coupling Layers

- Inverse and Jacobian determinant still ok to compute because of the following:

$$(f_b \circ f_a)^{-1} = f_a^{-1} \circ f_b^{-1}, \tag{8}$$

$$\frac{\partial (f_b \circ f_a)}{\partial x_a^T}(x_a) = \frac{\partial f_a}{x_a^T}(x_a) \frac{\partial f_b}{x_b^T}(x_b = f_a(x_a)), \tag{9}$$

$$\det(AB) = \det(A)\det(B). \tag{10}$$

Multi-Scale Architecture

- Implement a multi-scale architecture using *squeezing*.
  $\rightarrow (s, s, c) \rightarrow (\frac{s}{2}, \frac{s}{2}, 4c)$.
  $\rightarrow$ Trade spatial size for number of channels.

Multi-Scale Architecture

- Implement a multi-scale architecture using *squeezing*.
  $\rightarrow (s, s, c) \rightarrow (\frac{s}{2}, \frac{s}{2}, 4c)$.
  $\rightarrow$ Trade spatial size for number of channels.
- At each scale:
  $\rightarrow$ Apply 3 coupling layers with alternating checkerboard masks.
  $\rightarrow$ Squeeze.
  $\rightarrow$ Apply 3 coupling layers with alternating channel-wise masks.
  $\rightarrow$ Only apply 4 coupling layers with alternating checkerboard masks in the final scale.

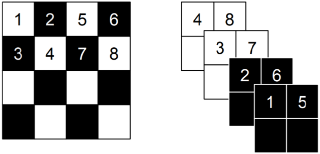


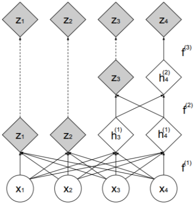
Figure 3: Masking schemes for affine coupling layers. On the left, a spatial checkerboard pattern mask. On the right, a channel-wise masking. The squeezing operation reduces the $4 \times 4 \times 1$ tensor (on the left) into a $2 \times 2 \times 4$ tensor (on the right). Before the squeezing operation, a checkerboard pattern is used for coupling layers while a channel-wise masking pattern is used afterward.

Multi-Scale Architecture

- It is a pain to propagate all D dimensions across all layers.
  $\rightarrow$ Memory.
  $\rightarrow$ Computational cost.
  $\rightarrow$ Number of trainable parameters.

Multi-Scale Architecture

- Factor out half the dimensions at regular intervals.
- Gaussianize such units, concatenate to obtain final output.
  - → Distributes the loss throughout the network.
  - → Learns local, fine-grained features.



(b) Factoring out variables. At each step, half the variables are directly modeled as Gaussians, while the other half undergo further transformation.

Results and Samples

(Show paper.)

Summary

- Use *change of variable* to express likelihood of data.
- Use *coupling layers* to define bijection between data and latent spaces.
- Train using *maximum likelihood*.
- Can perform exact and efficient
  - inference,
  - sampling,

References

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio.
"Density estimation using Real NVP."
arXiv preprint arXiv:1605.08803 (2016).

Oord, Aaron van den, Nal Kalchbrenner, and Koray Kavukcuoglu.
"Pixel recurrent neural networks."
arXiv preprint arXiv:1601.06759 (2016).

The End